Asian Journal of Applied Science and Technology (AJAST)
Volume 10, Issue 2, Pages 108-125, April-June 2026

Evaluating the Architectural Impact of Monolithic and Microservices Styles on Operational Efficiency in Modern Software Systems

Theophilus Bamise Ajala[1*], Ayorinde Oduroye[2], Idris Dolapo Ayoade[3], Abiodun Richard Agboola[4] & Idris Sunday Oladosu[5]

[1,2,4]Department of Computer Science, College of Computing and Information Systems, Caleb University, KM 15, Ikorodu-Itoikin Road, Imota, P.M.B. 21238, Lagos State, Nigeria. [3]Department of Computer Science, School of Technology, The Polytechnic, Ibadan, Polyibadan, Sango-Eleyele Express Road, P.M.B. 22, Oyo State, Nigeria. [5]Computer Science Department, College of Computing Sciences, Federal University of Agriculture, Abeokuta, Alabata Road, P.M.B. 2240, Abeokuta, Ogun State, Nigeria. Corresponding Author Email: ajalabamise@gmail.com[*]

DOI: https://doi.org/10.38177/ajast.2026.10209

Copyright © 2026 Theophilus Bamise Ajala et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Article Received: 09 April 2026 *Article Accepted: 17 June 2026* *Article Published: 19 June 2026*

ABSTRACT

Technology has revolutionized software development, forcing developers and operations to collaborate to design systems that are resilient, scalable, and efficient. This paper analyzes the two software architectural approaches namely Monolithic Architecture vs Microservices Architecture when considering scalable software development, we put the following into consideration: empirical trade-offs, scalability consideration, organizational implications, and development complexity. Some case studies were considered such as Banking sector, media company and a web-based travel application. In monolithic architecture, the application is constructed using one module only. The user interface, business logic, and data access are combined using the same coding framework. As all modules will be built using one framework, implementation and deployment become easier. Monolithic systems might struggle as applications grow. It is also associated with the following: In terms of cost, it is low, leading to fastest and cheapest way to launch the first product. In the context of skillset, easier for new hires; skills are common. In the aspect of maintenance, high code debt makes maintenance very expensive. In comparison, microservices architecture divides an application into smaller services that carry out specific business tasks. This is accompanied by the following features: From a financial perspective, there are high costs required at the start of implementation due to new tool purchases. Regarding skills, it requires specialists who have experience working on complex systems. When it comes to maintenance, it involves lower costs and targeted updates. In order to address challenges related to scalability and high performance, the team needs to determine the architecture that will work for them to create robust and easily deployable systems.

Keywords: Software Architecture; Monolithic; Microservices; Scalability; Software Design; Technology; Software Systems; Operational Efficiency; Complexity; Autonomy; Organization Skillset.

1.0. Introduction

Technology has impacted the way software development is done, with development and operation teams being required to collaborate to create highly resilient and scalable software systems. Software architecture helps in the design and structuring of complex systems that include scalability, maintainability, and efficiency among others (Kothapalli et al., 2024). Software architecture can be defined as the framework upon which the functionality, scalability, and maintainability of software systems are based. Rapid advances in technology within the last ten years have significantly impacted software architecture. Software architecture influences the way software systems are structured and interact with each other (Agweny and Mbugua, 2025). The monolithic approach develops a software application in a manner where all features of the software application exist in a singular system. This means that the application user interface, business logic, and the data access layer coexist using the same code base. However, with growing software applications, monolithic architecture becomes hard to cope with. This happens because scalability would require the duplication of all components within the software. Such a move becomes inefficient in use of resources. Changes and updates could be cumbersome in such cases due to their impact on the whole system. Other problems may include dependencies between modules of projects, long integration period for developers, and long periods for deployment. Thus, to solve this problem, microservice architecture offers simplicity in management and maintenance of growing complexity.

In the past few years, software architecture experienced significant changes. Initially, software architecture mainly applied monolithic architecture that comprises all the features in one unified software. However, the rise in complexity and size of features led to a number of issues with scalability, adaptability, and maintenance. In order to overcome such problems, researchers attempted to apply alternative architectures like service-oriented architecture (SOA). The latter involves separation of software into various reusable modules whose interaction occurs through interfaces. Despite the benefits provided by SOA, its services rely on centralized management and ESB for interaction, which results in performance issues and rigidity, creating the risk of a single point of failure. Eventually, such changes resulted in development of microservice architecture (Hassan et al., 2025). Unlike SOA, microservices architecture divides an application into smaller, deployable services implementing particular business functionalities. The modularity of microservices helps to develop, deploy, and scale them individually, making it possible to allocate resources only where necessary. Furthermore, teams can use various frameworks for development of each service separately, increasing their potential and flexibility. Nevertheless, it becomes more difficult to coordinate communication between different services and ensure their integration and synchronization. Consequently, it is crucial to monitor and orchestrate (Kamisetty et al., 2023).

Firstly, the problem associated with the usage of microservices is concerned with the need for extra effort in implementing and scaling every microservice to the Cloud. Also, organizations that employ microservices may utilize various types of automation software like DevOps, Docker, Chef, Puppet, and automatic scaling. Such implementation helps to save money and time. Nevertheless, this process is followed by further development, migration, and integration. Hence, agility, independent growth, and scalability require significant investments that should be considered when selecting microservices as an architectural style. Furthermore, the orchestration of microservices in production is another challenge (Tapia et al., 2020).

The choice between the usage of monolithic and microservices architecture depends on many factors including application's size, complexity, teams' organization, and future scalability. Simple application development and deployment make monolithic architectures more preferable for small-size applications. However, complex application with considerable scalability and flexibility may be designed using microservices architecture. The migration of microservices also requires well-prepared actions to solve some challenges connected with deployment complexity and microservices orchestration (Kamisetty et al., 2023).

This paper analyzes some of the recent advancements made in software architectures during the last decade using scholarly sources, industry reports, and case studies. The analysis will focus on the following Monolithic vs. Microservices Architectures since Software architecture has undergone many changes from 2016 through 2026 due to technological advancement and business needs, therefore, Organizations need to select their software architectures depending on their requirements in terms of performance, security, and scalability. The paper will analyze their advantages and disadvantages and also identify when each software architecture can be effectively used. There is controversy concerning these two types of architectures; that is, while monolithic architectures have a single codebase which makes them easier to develop and maintain, they experience challenges when scaling up and adapting as the application grows, however, microservices architectures promote scalability and agility by decomposing applications into independently deployable services.

The paper highlights the significance of having a systematic evaluation of both architectural designs to support software designing decisions. It becomes increasingly vital to understand the drawbacks associated with both architectural designs as companies seek scalable designs capable of coping with growing user base and complexities of features. In an attempt to assist software developers, architects, students, and practitioners in making well-informed and sound decisions about scalability and various other characteristics including maintainability, performance, and development efficiency. This paper concentrates on the comparison of architecture, performance, operational challenges, and implications. The literature review is carried out using peer-reviewed journals, conference papers, and credible internet sources.

1.1. Statement of the Problem

There is a need for more effective separation of concerns in the development of novel software architectures. This is why software architecture designers and implementers have developed different design approaches that can help software become more efficient with resources and composition of functions (Nayim et al., 2023) throughout the software phases. While some architectural designs group deployment of its components into one level, others divide it into multiple levels or tiers. Even though there is a considerable number of studies focusing on either monolithic or microservices architecture, the lack of systematic analysis across a broad range of application contexts and operational needs persists. Existing studies usually limit themselves to investigating particular issues related to monolithic or microservices architectures. Therefore, there is inadequate empirical information for the guidance of software architects and organizations concerning the selection of the right software architecture approach for different organizational settings. There is a paucity of empirical information on the software architecture of locally-based technology-focused firms in Nigeria. Although there have been many investigations into globally-based firms in literature, there are very few investigations that are dedicated to analyzing the software architecture of organizations such as Moniepoint operating in Nigeria. These organizations run complex software systems that serve millions of people. They experience a range of problems relating to infrastructure, transaction processing, and availability of services, among others. It is necessary to fill this empirical information gap to find out how these software architectures work in such a setting and hence the rationale behind this research investigation.

1.2. Study Objectives

This paper is guided with the following objectives:

1) To study the principles and characteristics of monolithic and microservice architectures.
2) To examine the impact of monolithic and microservice architecture approaches on efficiency operations within modern software systems.
3) To compare the ability of monolithic and microservices architectures to scale up when faced with different loads of applications.
4) To consider the organization-level consequences of choosing monolithic and microservice architectural styles or designs.

5) To assess the technical difficulties that can arise from the use of monolithic and microservice architectures.

6) To offer suggestions to software designers and developers regarding choosing one architectural design over another.

2.0. Literature Review

Overview of Monolithic Architecture and Microservices Architecture

The following section delves deeper into the overview of both monolithic and microservices architectures.

2.1. Monolithic Architecture

One of the earliest software architectures is monolithic architecture. In this case, the main idea of the design is to develop the application which has all necessary components. Each of the components depends on others and in many instances, they do not work separately and even cannot be compiled. Monolith application contains lots of different libraries. On the other hand, when the companies grow, monoliths are growing as well. At the same time, all the classes, functions and namespaces are stored there. There are several advantages in using this approach, because all processes related to a request will happen in a single process. One more distinctive feature of monolith is that it has huge code base that consists of different interdependent components. Therefore, when some changes should be applied to one of them, the entire application needs to be deployed. A microservices architecture is an architecture of backend applications, in which several parts communicate and interact with one another and carry out various functions independently of others; each part is referred to as a service. The monolithic architecture is employed in applications that operate with limited amounts of data where only one programming language or database is used. This makes it impossible for large organizations that have numerous employees to work on a product effectively (Benavente et al., 2022). Monolithic architecture is an approach to designing software where the entire application is built as a single connected system. In this method, all the various components of the program are combined in a single whole with a shared code base. The whole system works as a connected system, managing all business processes in one large network. In case of any changes or upgrades, they are required to be made to the whole code base (Liji and Renjith, 2025).

Figure 1. Monolithic Architecture (Liji and Renjith, 2025).

2.1.1. Why Monolithic

Changes from one style of architecture to another occur as a result of certain stages that may arise while applying the architectural style. Architectural patterns of enterprise software show that monolithic architectures have certain

benefits as far as ease of development and consistency of operations are concerned for applications with well-defined boundaries and relatively modest levels of complexity. Applying monolithic architectures usually occurs within certain patterns used by enterprises, such as Model-View Controller (MVC), and without sacrificing simplicity of deployment (Kambhammettu, 2025). However, modifications made to any of the components will require re-deployment of the whole application (Kambhammettu, 2025). According to Marieska et al. (2025), scaling the system for increased traffic entails scaling of the whole application, and this is not effective. In addition, a problem arising in any component may affect the whole application's performance.

2.2. Microservice Architecture

The microservice architecture, which is alternatively known as the microservices, is entirely different from the others. In this case, you will build your application using a set of individual services communicating with each other either via APIs or through a message broker. Each of the microservices is a little software having its architectural framework with business logic and a database that serves a definite purpose (Liji and Renjith, 2025).

The microservice architecture pattern is now crucial because it is characterized by its independence and decentralization. There have been many uses of microservice patterns recently, especially among industries. Major firms such as Netflix, Amazon, and Spotify have also started using microservices, while others have switched from traditional architectures to microservices. The firms were hoping to use the advantages of microservices in order to make their systems capable of addressing existing issues and improving their quality and making software maintenance easy (Su et al., 2024). The business logic has been compartmentalized into minimalistic, specialized self-contained services, known as microservice architecture. Every architectural service has the responsibility to fulfill a specific business objective. The microservice architecture can be equated to a Lego structure which has been fragmented into several pieces. APIs or Application Programming Interfaces ensure that the pieces of the structure communicate with each other (Elgheriani and Ahmed, 2022).

Microservices refer to a form of backend server applications; microservices can be used to divide into multiple units and create communication between them. In doing so, we create a service and exchange data among themselves to perform functions independently of other services. Using such technology, we have the opportunity to develop software that includes different programming languages or database (Benavente et al., 2022). Undoubtedly, the most prominent advantage of MS is their outstanding ability to support large numbers of users/orders, called horizontal scaling (Batyrbekova, 2025). But the issue of the scalability of MS is critical for current software systems but poses many difficulties due to their nature (Gurung et al., 2025).

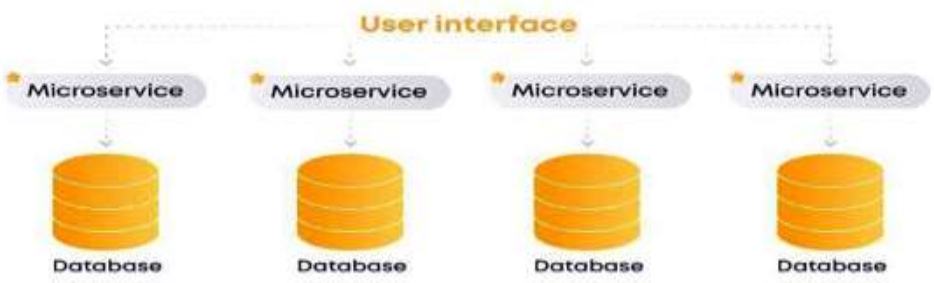

Figure 2. Microservice Architecture (Liji and Renjith, 2025).

The architecture pattern of microservices breaks down an application into loosely coupled modules that have their own storage, define well-crafted interfaces, and communicate using lightweight communication methods or well-designed APIs. This arrangement of the components ensures autonomy in terms of deployment and scalability. Independence is achieved by implementing the technology stack in accordance with the needs of individual services rather than implementing one single technology stack throughout the application due to decentralization and data storage of individual services. There is a significant correlation between the use of microservices architecture and DevOps methodologies because of the additional complexities introduced by distributed systems (Kambhammettu, 2025).

2.2.1. Why Microservice

Deployment of each microservice occurs individually No need to deploy all the application every time there are changes, and for complex applications, continuous deployment is possible. The development team of that particular microservice will have Microservice developed. Breaking down an application will ensure that developing the application is much simpler. Adopting new technology becomes easy because the developer is allowed to use whatever technology he deems appropriate for his particular microservice without the restrictions set at the beginning of the development process (Elgheriani and Ahmed, 2022).

Table 1. Differences between Monolithic and Microservice Architecture (Elgheriani and Ahmed, 2022).

Monolithic architecture	**Microservices architecture**
Service components that are tightly coupled	Service components are loosely coupled
Takes more time to build	Takes little time to develop
Dependent upon each other; if one fails, the whole system will fail	No matter what service fails, it will be fixed independently, without having an effect on the application
Distribution needs fewer resources since there is a centralized approach to dependencies.	Team works independently
It becomes quite hard to deliver applications continuously and quickly, since the process needs to start afresh every single time	The entire team makes lots of innovations, enabling fast development and deployment
Less scalable	Greatly scalable
Challenges in communicating among various teams	The communication in the team is efficient and understandable
Project oriented	Product oriented
Only one technology can be used at once, while upgrading to new technologies is a challenge	A variety of technologies could be used by various services.

Based on the information given in the Table 1, in case an organization wishes to move from monolithic to microservices, then the following paragraph will answer what should be done. The initial step when moving towards microservices is finding out where the proper boundaries for the services lie. To achieve this, there is the need to study the monolithic architecture to come up with ways that functionalities can be isolated in separate services. There is also the need to think about the Domain Driven Design (DDD) (Duvvur, 2022). The DDD helps

to establish service boundaries. In DDD, software modules are created based on business domains or capabilities, meaning that services have their boundaries and do not overlap in terms of functionalities. For instance, an e-commerce system can consist of product management, order management, and customer management services (Duvvur, 2022).

2.3. Empirical Trade-Offs

The following summary provides a synthesized view of the key trade-offs found in the studies between monolithic (ML) and microservice (MS) architecture (Batyrbekova, 2025).

Technical: Scalability

MS: High. Superior at handling large user volumes.

ML: Low. Limited by one server and single point of failure.

Technical: Complexity

MS: Very High. Difficult to track, needs tools for many services.

ML: Low. Easy to track and manage data in one place.

Organizational: Autonomy

MS: High. Allows teams to make their own decisions quickly.

ML: Low. Shared code creates delays and requires permission from many teams.

Organizational: Skillset

MS: High Requirement. Needs experts in complex distributed systems.

ML: Low Requirement. Easier for new hires; skills are common.

Economic: Initial Cost

MS: High. Significant starting investment in new tools.

ML: Low. Fastest and cheapest way to launch the first product.

Economic: Long-Term TCO

MS: Lower (at scale). Cheaper maintenance and targeted updates.

ML: Higher (at scale). High code debt makes maintenance very expensive.

2.4. Comparative Analysis

Scalability is one aspect that needs consideration during software architecture design since it determines how well the application can handle increased demands for growth. Scalability outcomes are highly dependent on the chosen software architecture style, either microservice or monolithic.

2.4.1. Scalability Consideration

Monolithic architecture integrates all the different elements of a program into one coherent code base. Scaling through vertical approach means scaling up by increasing the current computing power of the existing system by

adding additional components like memory or processor. While this can be effective, there will be challenges like hardware limits and high costs that may not make it feasible. The main problem when dealing with monolith applications is scaling horizontally since there are tight couplings between its components. As a result, replication of the entire program would be required across multiple computers, causing inefficiency (Kamisetty et al., 2023).

The application is divided into independently scalable components called microservices, where each microservice is responsible for performing different business operations. This approach makes the system horizontally scalable, meaning that different components can scale out depending on their load to increase performance without affecting the entire system. If there is too much pressure on any component, many such components can be created to cope with the additional load and optimize resources. The benefit of this approach is not only scalability but also resilience and isolation of faults. Stability of the whole system is guaranteed by the fact that failure in one microservice will not affect others (Kamisetty et al., 2023).

For the business to expand and to maintain uninterrupted operations, the systems should be scalable. Scalability is defined as the capacity for the system to accommodate an increased load through the addition of more resources. In cases where a business supports 50,000 customers now but expects to expand to reach 700,000 or even more in the future, the system will need to be able to manage the increased load (Mhamankar and Piyarji, 2026).

Impact on scalability when deciding whether to use Monolithic or Microservice:

Resource Efficiency: While monoliths scale in an all-or-nothing approach, microservices help in achieving better resource efficiency by facilitating custom scaling of certain elements.

Agility in Development: The development of microservices involves self-contained development and deployment, allowing developers to make changes rapidly, thus achieving better scalability.

Increased Complexity: While microservices help improve scalability, they are complex in terms of service coordination, deployment, and monitoring (Kamisetty et al., 2023).

2.4.2. Organizational Implications

The choice of architecture is very important in determining the composition of teams and their communication patterns. While monoliths usually match centralized teams, microservices allow implementing Conway's Law by forming autonomous teams for each business capability that need more complex organizational coordination mechanisms. The fit between different types of architectures and the organization's structures is one example of how Conway's Law works, as it says that system architectures are a reflection of the communication patterns of the organizations that build them. In monolithic architectures, centralized teams of specialists are usually used, making the implementation process uniform but resulting in more coordination when working on larger teams. In a centralized approach, decisions will most likely be made from the center, with the architecture being managed consistently throughout the whole application. Microservice architecture provides autonomy to the teams, using distinct service boundaries and ownership for certain business capabilities. (Kambhammettu, 2025). The architectural choices that are made in the very beginning when designing enterprise software systems have far-reaching effects on the performance, scalability, maintenance, and speed with which an organization responds

to changing needs (Bellundagi, 2026). From the perspective of managing human resources, Gao and Liu (2025) opined that given the context that today's human resource systems have transformed from being record-keeping systems to being participative and even smart systems, such systems should be able to quickly adapt to changing demands, including those involving telecommuting and flexible benefits. The advent of microservices architecture provides an avenue towards modernization. At the same time, organizations are faced with the dilemma of refactoring their systems or directly applying microservices when designing new ones. In addition, the adoption of monolithic architecture which will ensure relatively lower initial costs would be ideal for businesses which run without many challenges such as start-up firms (such as small and medium sized firms) while the upfront cost of adopting microservice architecture for large firms will prove economically feasible in the long term (Gao and Liu, 2025).

The studies prove beyond doubt that MS is more suitable when it comes to organizing people. The size of the units is not big, thus smaller groups can independently take decision-making roles and operate independently. In this way, the entire process gets faster. However, as the number of developers grows in a Monolith, the development process inevitably becomes extremely slow. The tight connection of the codes implies that each change should be discussed and approved by numerous groups.

2.4.3. Development Complexity

In (Azevedo et al., 2026) a good architecture acts as a solid foundation for any system, giving an organization a good starting point for further development. Sometimes, when less resources and efforts are put into this element, it will cause quality problems later on for the development of the product.

In terms of design, monolithic systems are easy to debug, test, and deploy. On the other hand, microservices add complexity to software due to service boundaries, distributed communication, and necessary infrastructure (Kambhammettu, 2025).

The difference in complexity in architectural styles arises in implementation, debugging, and testing. Monolithic architecture simplifies the complexity of development due to easy tooling needs and direct function calls. The monolith approach ensures that debugging is simple due to consistent contexts in which the components interact within the process. In such cases, developers can concentrate on the application logic instead of infrastructure issues while coding the application in the early stages. This simplification of the development process will make it easier to implement features in the initial stages of development. (Kambhammettu, 2025).

The complexity in the development of software is increased by the use of microservices architecture because of the service borders and the communication mechanism. The distribution of the architecture adds complications in debugging because of the transactions crossing several different services and the need for an advanced tracing system. There is also the need for testing that extends into contract testing and integration. These are some of the challenges that require investment in automation systems in the development process. (Kambhammettu, 2025).

In the aspect of developing real time systems, Bhandari (2025) argue that the selection of correct architectural patterns of the real-time application is not an easy task and cannot be decided based solely on technological

considerations. In a real-time system, there are certain time constraints, where processing delay carries some commercial and usability meaning. The monolithic pattern makes possible the integration and reduced communication overhead, which are essential in some cases of real-time processing. In addition to that, the microservices architecture ensures isolation from failures and scalability, a system whose processing needs vary with respect to different functionalities. The selection will have a lasting impact on the further development of the system (Bhandari, 2025).

2.5. Case Studies

This paper will review some case studies to justify the strengths and weakness of these architectural styles.

2.5.1. Case study of Moniepoint

In the current dynamic world of software engineering, maintaining competitiveness and catering to the needs of users has never been more important. Microservices provide an agile way for designing robust applications. The Monolithic Architecture provides a simple and effective way to manage code for applications. By packing everything into one executable, the process of developing, testing, and deploying code becomes much simpler. Nevertheless, the simplicity of this architecture can prove to be its weakness as the application becomes bigger and more complicated. In case of Monolithic Architecture, in order to make any changes in the application, the entire application would need to be deployed again. While to reduce the time of downtime several methods like Canary Deployments can be used (Duru, 2023). According to Duru (2023) “In Microservices Architecture, which is the framework we use heavily at Moniepoint, our approach is somewhat different.” The application is divided into multiple loosely coupled services that interact using the API. Each service can be separately built, tested, and deployed. Hence, using Microservices makes it possible for developers to work with simpler components.

Microservices provide better scalability, high availability, and fault tolerance when compared to other options such as Monolithic architecture. While Microservices are said to be an ideal approach when it comes to creating highly scalable applications, Monolith should not be ignored either. However, in some cases, a well-designed Monolith might offer you all the same advantages as those offered by Microservices. In that case, we refer to it as a Modular Monolith or Modulith. A Modular Monolith allows you to have all the advantages of a Monolith along with the ease of modularity in architecture while still preserving the code quality and its maintainability. All the modules in a Modular Monolith are separated from one another; hence, it results in decoupled modules that can easily be maintained. Thus, depending upon your specific requirements and needs, you may want to consider both approaches (Duru, 2023).

2.5.2. Case Study of Danske Bank

The positive effects of microservices concerning scalability have been highlighted through empirical studies. For example, scalability became better after redesigning a monolithic architecture as microservices based on a case study that investigated the migration of a critical system at Danske Bank. The results showed that, apart from utilizing less memory and CPU power, the microservices architecture led to better performance in regard to modular software characteristics (Kamisetty et al., 2023).

Table 2. Performance Benchmarking Results Table (Kamisetty et al., 2023).

Test scenario	Monolithic response time	Microservices response time	Monolithic throughput	Microservices throughput
Single user load	200ms	180ms	100 req/s	110 req/s
100 concurrent users	2s	1.8s	50 req/s	55 req/s
1000 concurrent users	20s	15s	10 req/s	12 req/s

Table 2 refers to the tabular presentation of the data used to evaluate and compare the performance of different systems or products using set standards or criteria. Such tables can play an important role in assessing efficiency, pinpointing problems for improvement, and making evidence-based decisions. The response time grew in the case of monolithic systems, whereas the response time shrank in the case of microservices. Scalability is an important element when designing software architecture because it impacts user satisfaction, performance, and the ability of the business to grow. Monolithic systems can suffice in applications that do not need high levels of scalability because such systems are relatively easy to develop (Kothapalli, 2021). On the other hand, microservices systems serve better than monolithic systems in applications that require high scalability and significant growth potential. Microservices provide a way of resource allocation tailored to the application in order to meet its increasing demands while ensuring system coherence by enabling independent scaling of services. The choice between microservices and monolithic systems should depend on the level of expected scalability. (Kamisetty et al., 2023).

2.5.3. Case Study of Four Media Companies

As indicated in the works of Su et al. (2024), there are five primary reasons behind reverting back to monolithic architecture from microservice architecture, which are described in the Table below:

Table 3. Reasons for cases switch back to monolith

Reason	Case
Cost	Istio control plane, Prime video monitoring service, Segment, In Vision
Complexity	Istio control plane, Segment, In Vision
Scalability	Istio control plane, Prime video monitoring service, Segment
Performance	Segment
Organization	Istio control plane, In Vision

Four companies represent the case under this case study. Each of the reason is briefly explained in the paragraph below:

Cost is one of the most common reasons why businesses revert from microservices to monoliths. In the Istio control plane, for instance, there was an increase in marginal cost in the independently scalable components due to the adoption of the microservices architectural pattern, hence creating little to no value. Amazon Prime Video also stated that it saved over 90% on infrastructure costs by shifting from microservices to monolithic architectures.

Among the most critical factors causing enterprises to reverse their decision and return to the monolith from the microservice, complexity ranks as the most notable. From the Istio control plane perspective, we observed that the adoption of the microservice design resulted in complexity from its content, management, and technology. To start, its various planes had various languages for programming. Moreover, different modules were handled by different teams. In the process, however, complexity was added to the process.

Scalability is another issue where the ability of microservices to scale its components independently acts as one of the factors behind the migration of enterprises to the microservices architecture. But, here, it became a problem. XDS is a component responsible for cost calculation in Istio. These were low-cost components, which made isolation of microservices insignificant. As a result, Istio control plane was changed back to monolith, hence simplifying scalability issues.

With respect to Performance, Segment was the only company that mentioned performance in its list of reasons for going back to using the monolith out of the above-mentioned four companies, and this was one of the main reasons behind the decision of Segment to return to the monolith. The worst problem with this was the issue with the line head blocking. Head-of-line blocking occurred as a result of microservice architecture whereby a single failure on one of the destinations results in failures on all destinations.

The reason for organization was that there was a high level of complexity in the management of teams, which led to switching back to the monolith, especially Istio Control Plane and InVision. Although microservices provide for separate service management by different teams within Istio Control Plane, it may lead to confusion for the developers to be able to work efficiently.

InVision had a legacy team that was small in number, yet it owned numerous repositories, databases, programming languages, and so forth.

2.5.4. Case Study of Travel Application

The Software Company has put in place a travel solution, and this software has incorporated both architecture types for comparison purposes.

E-tickets make buying tickets easier, less risky, and more convenient to the customers since tickets can be accessed easily and instantly. E-tickets also connect to mobile applications and notify users of instant updates on their tickets. The online ticketing system was chosen in this case study because of the high volume of transactions that have to be executed within a short while. For example, when there is heavy tourist traffic in a country such as Dubai where e-ticketing is done as part of the process to secure hotel accommodation, the system needs to be able to handle many transactions quickly and effectively.

Application Architecture

Travel: Prices of flights shown and purchase of flight tickets.

Hotel: Displaying hotel rooms and booking a room.

Purchase: Giving the capability of making payments to users.

Identification: An identification service that provides sign-ups, log-ins, log-outs, etc.

Monolithic architecture in use

Having to work with so many route choices and a variety of transport types, an abundance of offers, promotions, scheduling, changing terminals, and seemingly endless journey combinations is an enormous engineering feat. The miracle that happened from clicking the search button to seeing the most appealing route options listed was initially taken care of by a single monolithic application, because at the time there were fewer requests, less data to process, smaller codebase, and fewer connections with third party services than there are today. As all of those numbers started growing very quickly in more ways than one, it became harder to develop and maintain. I'm sure everyone who works there knows exactly what came after – by using modern software architecture and technologies. All provider-related services were designed as monolithic applications because of simplicity reasons.

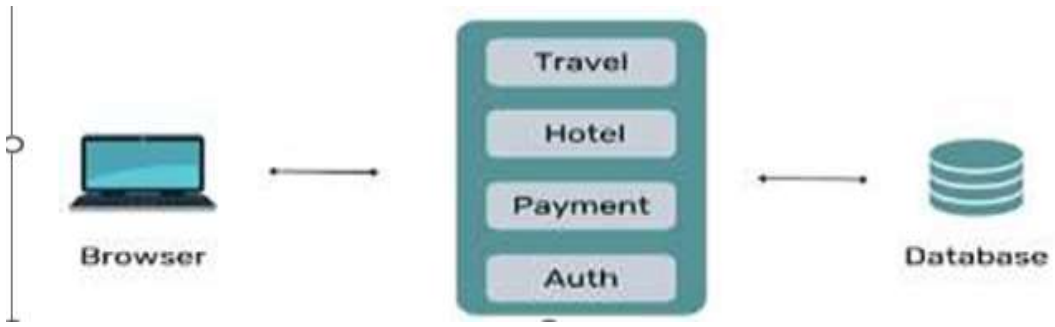

Figure 3. How Monolithic works on the Travel Application (Elgheriani and Ahmed, 2022)

Microservice architecture in use

With the increased complexity involved in the development process, it became harder for the software to start up and consumed more and more resources. Therefore, adding additional copies in the production environment became hard and not efficient in terms of CPU and memory usage. However, while it became tough to integrate all the service providers due to the difference in technology and poor technical knowledge, the new architectural base is required eventually.

Figure 4. How Microservice works on the Travel Application (Elgheriani and Ahmed, 2022).

3.0. Methodology of the Study

A qualitative literature review research methodology will be used in this study to assess the architectural framework that can help to enhance the operational efficiency in scalable software. There is no need for collecting primary data through respondents in this study since secondary data is utilized in assessing the architectural framework under

analysis. This means that the study involves collecting secondary data from credible sources such as scholarly journals, reports, documents, and studies in order to gain an insight into the merits, demerits, and scalability capabilities of monolithic architecture and microservices architecture. Secondary data is collected from credible academic and industrial sources. Literature for this study is collected from Google Scholar and other credible sources. Other relevant information is collected from technical documents prepared by technology firms and software developers. Relevant literature includes sources related to monolithic architecture, microservices architecture, scalability of software, system performance, operational efficiency, and software maintenance. The literature screening involved the identification of sources that have dealt with issues concerning the evaluation of software architecture, scalability issues, performance issues, implementation approaches, maintenance considerations, and experiences with organization adoption. This was based on the relevancy, credibility, quality of publication, and usefulness of sources to comprehend the differences between monolithic and microservices architectural systems. Sources unrelated to the subject were not included in the study. The data collection entailed performing a systematic search using relevant keywords like "Monolithic Architecture," "Microservices Architecture," "Software Scalability," "Operational Efficiency," "Distributed Systems," and "Software Architecture Evaluation." The comparative approach was used to evaluate each architectural system and its role in facilitating operational efficiency and scalability.

4.0. Results and Discussion

From the findings in Kamisetty et al. (2023) regarding Danske Bank, it was found that there were considerable differences in each scenario where the numbers of concurrent users varied, especially those with 100 concurrent users. There were evident signs of variations and spikes in the figures collected. When it comes to the performance of response time, the performance of microservices was better where there were 100 concurrent users.

On the basis of the outcome of the case study discussed in Su et al. (2024) concerning four firms. Cost and complexity have been one of the major factors that lead companies to switch from microservice back to monolith architectures. In particular, this is because of the limited availability of computing resources, especially that of CPU. The effects of such fluctuations become increasingly noticeable in a microservices environment. The reason being that this architecture requires a higher number of resources compared to a monolithic architecture because it can run more than one instance, consisting of services, database, and message broker.

According to the results from the case study provided by Elgheriani and Ahmed (2022), regarding Travel Application, Monolith will be incredibly fast when there is no traffic, while MS will have a significant advantage when it comes to scaling due to its ability to scale only those parts that are experiencing high traffic. Stretching capability is crucial for very large applications. However, Monolith will work faster and more efficiently in terms of managing the data, since all the data are in one place. This fact guarantees its accuracy. On the contrary, MS complicates the situation, since distributed transactions make it difficult to maintain data integrity.

The conclusions drawn from this experiment align with one of the conclusions from the experiment by (Gurung et al., 2025), which relates to scalability issues that result in degraded performance for applications within microservices architecture. The microservices architecture needs more resources than the monolithic architecture

since it is capable of running multiple instances at once. The results of performance and scalability for ecommerce and web applications indicate that the microservices architecture may perform better than the monolithic architecture when faced with high traffic conditions.

The analysis of the literature has been used to make evidence-based suggestions and recommendations in relation to software developers, software architects, researchers, and technology organizations. Through the results obtained, the study offers insight into understanding how various software architectures can be scaled in order to meet business requirements, increase efficiency, and ensure that the resulting software architecture is sustainable and reliable. The study is useful in guiding decision-making in terms of selecting suitable architectural styles depending on the size of the application, organizational needs, and expected future growth.

Lastly, the architectural scene keeps evolving as organizations operate between well-established architecture types and new methods of architectural implementation. This paper on the different software application architectures highlights the fact that there needs to be an important consideration of factors before deciding between monolithic architecture and microservices architecture. There is usually some sort of point when transitioning from one architectural type to another due to some reason experienced in application. Regarding monolithic and microservices: regarding integration of components, it is tightly coupled (the components are integrated together) and it is loosely coupled; communication method, it uses direct method and shared libraries and it uses APIs and different libraries; data management, it uses centralized data storage or codebase and it uses decentralized data storage. Deployment, regarding change, redeployment of the whole application is required and redeployment of each service separately is required.

5.0. Conclusion

Monolithic architecture implementations adopt the MVC pattern, but Microservice implementations align themselves with the DevOps approach, which deals with the additional complexity that arises. Software development appears to be increasing due to the need for businesses to operate online. The idea is that applications that can assist businesses in becoming efficient and growing are being created. This means that software companies always look for technological ways that will make their applications agile and scalable. The modern software architectures involve two popular architectural designs when it comes to software development, meaning that the decision to adopt Monolithic or Microservice architecture depends on the particular project as well as the company itself. Therefore, it has become vital to analyze the effects of adopting certain types of architecture in the wake of increased digitalization within organizations. The decision will determine whether to adopt the monolithic, the microservices, or the move from one type of architecture to another as there is no architecture that suits all situations. It is clear that monolithic architecture will still be relevant in simplistic scenarios, while the microservices will offer more flexibility in case of complexity where maturity of operations is achieved. With regard to future advancements, the industry has embarked on developing hybrid architectures from both monolithic and microservices such as modular monolith architecture (MMA).

6.0. Future Recommendations

The following suggestions are offered:

1. Hybrid Architecture Models Should Be Investigated: Further studies should be undertaken to examine hybrid architecture models, including Modular Monolith Architecture (MMA), which merge simplicity with flexibility and scalability.
2. Empirical Analysis in Nigerian Technology Enterprises: Further studies should analyze software architectures employed by Nigerian technology enterprises to gain locally relevant results.
3. Impact of New Technologies on Architectures: Future research should assess the impact of emerging technologies like cloud computing, containers, DevOps, Kubernetes, serverless computing, and AI on the scalability of monolithic vs. microservices architecture.
4. Security and Reliability Issues: Further studies should consider the issues of security, resilience and fault recovery, and cybersecurity in monolithic and microservices architectures.
5. Development of Decision-Making Frameworks: Further analysis should be aimed at designing decision support frameworks that would help software architects to choose architecture based on application size and other criteria.

Declarations

Source of Funding

This study did not receive grant from any funding agencies in the public, commercial, or not-for-profit sectors.

Competing Interests Statement

The authors declare no competing professional, personal or financial interests.

Consent for Publication

The authors declare that they consented to the publication of this article.

Authors' Contributions

All the authors made an equal contribution from conceptual design, data collection, drafting the article to revision of the article. All the authors have read and approved the final copy of the manuscript.

References

[1] Agwenyi, C.A., & Mbugua, S.M. (2025). Trends in software architecture designs: Evolution and current state. International Journal of Innovative Science and Research Technology, 10(3). https://doi.org/10.38124/ijisrt/25mar1311.

[2] Azevedo, R.J.L., Ribeiro, O., Lima, B.T.S., & Ferreira, L.G.M. (2026). Assessing monolith and microservices architectures. Procedia Computer Science, 278: 767–774. https://doi.org/10.1016/j.procs.2026.03.049.

[3] Batyrbekova, S. (2025). An empirical comparison of microservice and monolithic architectures in software development.

[4] Bellundagi, M. (2026). A comparative study of monolithic vs microservices architecture AI-based performance analytics. International Journal of Machine Learning and Artificial Intelligence.

[5] Benavente, V., Yantas, L., Moscol, I., Rodriguez, C., Inquilla, R., & Pomachagua, Y. (2022). Comparative analysis of microservices and monolithic architecture. https://doi.org/10.1109/cicn56167.2022.10008275.

[6] Bhandari, M. (2025). Microservices vs. monolithic architectures in real-time distributed systems: A comparative analysis. Journal of Information Systems Engineering and Management, 10(62s). https://doi.org/10.52783/jisem.v10i62s.13614.

[7] Duru, D. (2023). Microservice architectures and design patterns [Online]. Website. https://moniepoint.com/blog/microservice-architectures-and-design-patterns.

[8] Duvvur, V. (2022). Transitioning from monolithic to microservices architecture: Strategies, challenges, and opportunities. Journal of Scientific and Engineering Research, 9(3): 212–215.

[9] Elgheriani, N.S., & Ahmed, N.A.S. (2022). Microservices vs. monolithic architectures: The differential structure between two architectures. International Journal of Applied Sciences and Technology. https://doi.org/10.47832/2717-8234.12.47.

[10] Gao, Z., & Liu, Z. (2025). Microservices architecture versus monolithic architecture: Technical trade-offs and economic analysis of modularising human resources systems. International Journal of Multidisciplinary Research (IJMR). https://doi.org/10.65231/ijmr.v2i1.124.

[11] Gurung, N., Shrestha, S., & Chulyadyo, R. (2025). Scalability in microservices: A systematic literature review. Journal of Computer Science and Technology, 25(2).

[12] Hassan, H., Abdel-Fattah, M.A., & Mohamed, W. (2025). A pattern-based framework for automated migration of monolithic applications to microservices. Big Data and Cognitive Computing, 9(10): 253. https://doi.org/10.3390/bdcc9100253.

[13] Kambhammettu, A.P. (2025). Monolithic versus microservice architectures: A comparative analysis for enterprise applications. European Journal of Computer Science and Information Technology, 13(51): 65–75. https://doi.org/10.37745/ejcsit.2013/vol13n56575.

[14] Kamisetty, A., Narsinar, D., Rodriguez, M., Kothapalli, S., & Gummadi, J.C.S. (2023). Microservices vs. monoliths: Comparative analysis for scalable software architecture design. Engineering International, 1(2).

[15] Kothapalli, S., Nizamuddin, M., Talla, R.R., & Gummadi, J.C.S. (2024). DevOps and software architecture: Bridging the gap between development and operations.

[16] Liji, T.P., & Renjith, R. (2025). A comparative analysis of monolithic and microservices architecture. IJARSAI International Journal of Advanced Research Scientific Analysis and Inferences, 1(1).

[17] Marieska, M.D., Yunanta, A., Aulia, H., Utami, A.S., & Rizqie, M.Q. (2025). Performance comparison of monolithic and microservices architectures in handling high-volume transactions. JURNAL RESTI (Rekayasa Sistem dan Teknologi Informasi), 9(3): 594–600. https://doi.org/10.29207/resti.v9i3.6183.

[18] Mhamankar, V.D., & Piyarji, A.I. (2026). Comparative study of scaling monolithic and microservices. International Journal of Engineering Research & Technology (IJERT), 15(2).

[19] Papadhopulli, I., & Alijia, A. (2024). Comparison between monolithic and microservices architecture in web applications built with Java. International Scientific Journal "Security and Future".

[20] Su, R., Li, X., & Taibi, D. (2024). From microservice to monolith: A multivocal literature review. Electronics, 13: 1452. https://doi.org/10.3390/electronics13081452.

[21] Tapia, F., Mora, M.Á., Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020). From monolithic systems to microservices: A comparative study of performance. Applied Sciences, 10(17): 5797. https://doi.org/10.3390/app10175797.